

Paraunitary Matrices, Entropy, Algebraic Condition Number and Fourier Computation*

Nir Ailon¹

1 Department of Computer Science
Technion Israel Institute of Technology
Haifa, Israel
nailon@cs.technion.ac.il

Abstract

The Fourier Transform is one of the most important linear transformations used in science and engineering. Cooley and Tukey's Fast Fourier Transform (FFT) from 1964 is a method for computing this transformation in time $O(n \log n)$. From a lower bound perspective, relatively little is known. Ailon shows in 2013 an $\Omega(n \log n)$ bound for computing the normalized Fourier Transform assuming only unitary operations on two coordinates are allowed at each step, and no extra memory is allowed. In 2014, Ailon then improved the result to show that, in a κ -well conditioned computation, Fourier computation can be sped up by no more than $O(\kappa)$. The main conjecture is that Ailon's result can be exponentially improved, in the sense that κ -well condition cannot admit $\omega(\log \kappa)$ speedup.

The main result here is that 'algebraic' κ -well condition admits no more than $O(\sqrt{\kappa})$ speedup. The definition of algebraic condition number is obtained by formally viewing multiplication by constants, as performed by the algorithm, as multiplication by indeterminates, giving rise to computation over polynomials. The algebraic condition number is related to the degree of these polynomials. Using the maximum modulus theorem from complex analysis, we show that algebraic condition number upper bounds standard condition number, and equals it in certain cases. Algebraic condition number is an interesting measure of numerical computation stability in its own right. Moreover, we believe that the approach of algebraic condition number has a good chance of establishing an algebraic version of the main conjecture.

1998 ACM Subject Classification F. Theory of Computation F.2.1 Numerical Algorithms and Problems

Keywords and phrases Fourier Transform, Paraunitary matrices, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The (discrete) normalized Fourier transform is a complex linear mapping sending an input $x \in \mathbb{C}^n$ to $y = Fx \in \mathbb{C}^n$, where F is an $n \times n$ unitary matrix defined by

$$F(k, \ell) = n^{-1/2} e^{-i2\pi k\ell/n}. \quad (1)$$

The Fast Fourier Transform (FFT) of Cooley and Tukey [6] is a method for computing the Fourier transform of a vector $x \in \mathbb{C}^n$ in time $O(n \log n)$ using a so called linear-algebraic algorithm. A linear-algebraic algorithm's state at each point in the computation is a vector

* This work was supported by a consolidator ERC grant INF-COMP-TRADEOFF



in \mathbb{C}^n , representing a linear transformation of the input. Each coordinate of the vector is a complex number stored in a memory location (or rather, two real numbers, one representing the real part and the other the imaginary part). The next state is obtained from the current one by a simple linear algebraic transformation. We will define this precisely in what follows.

As for lower bounds, it is trivial that computing the Fourier Transform requires a linear number of steps. Papadimitriou proves in [10] that the graph structure of Cooley and Tukey's FFT algorithm is optimal for a version of the transformation over finite fields. The result, however, assumes that the algorithm can only multiply by roots of unity of order n , and cannot 'take advantage' of any algebraic identity involving these roots. It is not clear how to apply these results to computation over the reals, or over the complex fields. Winograd [11] was able to reduce by 20% the number of multiplications in Cooley and Tukey's FFT, without changing the number of additions. In [12] the same author shows that at least linearly many multiplications are needed for performing Cooley and Tukey's algorithm. We note that the separation between 'additions' and 'multiplications' should be done with care in the real/complex setting, due to a question of scaling, as we now explain. Cooley and Tukey's original algorithm was defined for the *unnormalized* transform $\sqrt{n}F$. A normalized version would need to scale down the final result by a factor of \sqrt{n} . We should be, at the very least, conscious of the issues arising when outputting numbers of magnitude $\Omega(\sqrt{n})$ larger than the input numbers. At first sight this seems to be hardly an issue when computing with 64 bit words, but for various optimization purposes and applications we might actually want to squeeze multiple numbers in one computer word, in which case this may be an issue. One way to deal with this is to normalize each addition in the original FFT by dividing by $\sqrt{2}$. But this introduces a new multiplication together with each addition, so the separation between multiplications and additions is an approach that relies on scale. In 1973, Morgenstern proved that if the modulus of constants used in an *unnormalized* Fourier transform algorithm are bounded by a global constant, then the number of steps required is at least $\Omega(n \log n)$. He used a potential function related to matrix determinant. However, this result is also scale dependent, because it does not provide lower bounds for the (normalized) F .

Ailon [1] considered a computational model allowing only 2×2 rotation gates acting on a pair of coordinates. He showed an $\Omega(n \log n)$ lower bound on the number of such gates required for the (normalized) Fourier transform. The proof was done by defining a potential function on the matrices $M^{(t)}$ defined by the transformation that takes the input to the machine state in step t . The potential function is simply the sum of Shannon entropy of the probability distributions defined by the squared modulus of elements in the matrix rows. (Due to orthogonality, each row, in fact, defines a probability distribution). That work raised the question of *why shouldn't it be possible to gain speed by 'escaping' the unitary group?* In [3] he added nonzero, constant multiplication gates acting on a single coordinate to the model. By extending the entropy function to a so-called quasi-entropy, so called because it is applied to possibly negative numbers, the approach allows proving that a speedup in computing *any* scaling of FFT *requires* either working with very large or very small numbers (asymptotically growing accuracy). The bounds are expressed using a quantitative relationship between time (number of gates) versus the notion of condition number of matrices. More precisely, it is shown that speedup of FFT by factor of b implies that at least one $M^{(t)}$ must be $\Omega(b)$ -ill-conditioned. In [2], Ailon further showed that speedup implies ill-condition at *many* points in the computation, affecting independent information in the input. The work herein provides another result in this thread.

Ailon's results [1, 3, 2] are not satisfactory in the sense that the tradeoff between speed

and condition number is quite weak, and is believed to be suboptimal. For example, in the extreme case the results tells us that, if a linear time Fourier transform algorithm existed, then it would require a condition number of $\Omega(\log n)$ for most $M^{(t)}$'s. Such a restriction, though a step toward resolving the question of complexity of Fourier transform, is very mild, because (at least intuitively) such condition number can be accommodated using $O(\log \log n)$ -bit words. The net result is total running time (counting bit operations) of $O(n \log \log n)$. The main open question, informally stated, is as follows. We will restate it in a precise way in Section 6.

► **Conjecture 1.** [Informal] κ -Well conditioned computation allows only $O(\log(\kappa))$ speedup of FFT.

1.1 Our Contribution

This work defines a new *algebraic* notion of matrix condition number, which upper bounds the usual definition of condition number, and equals it in some cases. Using this new notion, we are able to show that κ -well conditioned computation allows only $O(\sqrt{\kappa})$ speedup of FFT, in other words, a quadratic improvement on the previous result (albeit with a weaker notion of condition). More importantly, we believe that an algebraic version of Conjecture 1 is achievable with respect to algebraic condition, and justify this belief at the end, in Section 6.

We also note that in the quantum world (see eg [9] and references therein), an $O(\log^2 n)$ -time algorithm is known using quantum gates, each unitary acting on only 4 entries. However, we do not deal with quantum computation here.

2 Fourier Transform Types

In theory and practice of engineering and computer science, there are two main types of Fourier transforms considered. The n -dimensional Discrete Fourier Transform (DFT), as defined in (1), is a complex unitary mapping defined by the characters of the Abelian group $\mathbb{Z}/n\mathbb{Z}$. The Walsh-Hadamard transform is a real orthogonal mapping defined by the characters of the n dimensional binary hypercube $(\mathbb{Z}/2\mathbb{Z})^n$. More precisely, for n an integer power of 2, the (i, j) 'th matrix element is defined as $\frac{1}{\sqrt{n}}(-1)^{\langle [i-1], [j-1] \rangle}$, where $\langle \cdot, \cdot \rangle$ is dot-product, and $[p]$ denotes the bit representation of the integer $p \in \{0, \dots, n-1\}$ as a vector of $\log n$ bits. Similarly to FFT for DFT, there is an $O(n \log n)$ algorithm for computing the Walsh-Hadamard transform of an input x .

Throughout, we will assume n is an integer power of 2 and will use F to denote either the n -Walsh-Hadamard, which is a real orthogonal transformation, or the real representation of the $(n/2)$ -DFT, which is a complex unitary transformation. By *real representation*, we mean the standard view of a complex number $a + ib$ as a column vector $(a, b)^T$ in two dimensional real space, and the multiplicative action $a + ib \mapsto (c + id)(a + ib)$ given by left-multiplication by the matrix $\begin{pmatrix} c & -d \\ d & c \end{pmatrix}$. In that way, a vector in n dimensional complex space embeds to a $2n$ dimensional real space, and a $n \times m$ complex matrix embeds as a $(2n) \times (2m)$ real matrix. This allows us to avoid defining a computational model over the complex field, because such computation can be emulated by reals anyway.

3 Computational Model and Notation

We remind the reader of the computational model discussed in [1, 3], which is a special case of the linear computational model. The machine state represents a vector in \mathbb{R}^ℓ for some

$\ell \geq n$, where initially it equals the input $x \in \mathbb{R}^n$ (with possible padding by zeroes, in case $\ell > n$). Each step (gate) is either a *rotation* or a *constant*.

A rotation applies a 2×2 orthogonal operator mapping a pair of machine state coordinates (rewriting the result of the mapping to the two coordinates). Note that a rotation includes the 2×2 mapping affecting a single coordinate, eg $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$. (Technically this is a reflection, but we refer to it as a *rotation gate* here.)

A constant gate multiplies a single machine state coordinate (rewriting the result) by a positive real constant. In case $\ell = n$ we say that we are in the in-place model. Any nonsingular linear mapping over \mathbb{R}^n can be decomposed into a sequence of rotation and constant gates in the in-place model, and hence our model is universal. Normalized FFT works in the in-place model, using rotations only. A restricted method for dealing with $\ell > n$ was developed in [3]. We focus in this work on the in-place model only.

Since both rotations and constants apply a linear transformation on the machine state, their composition is a linear transformation. If \mathcal{A}_n is an in-place algorithm for computing a linear mapping over \mathbb{R}^n , it is convenient to write it as $\mathcal{A}_n = (M^{(0)} = \text{Id}, M^{(1)}, \dots, M^{(m)})$ where m is the number of steps, $M^{(t)} \in \mathbb{R}^{n \times n}$ is the mapping that satisfies that for input $x \in \mathbb{R}^n$ (the initial machine state), $M^{(t)}x$ is the machine state after t steps. (Id is the identity matrix). The matrix $M^{(m)}$ is the target transformation, which will typically be F in our setting. For $t \in [m] := \{1, 2, \dots, m\}$, if the t 'th gate is a rotation, then $M^{(t)}$ defers from $M^{(t-1)}$ in at most two rows, and if the t 'th gate is a constant, then $M^{(t)}$ defers from $M^{(t-1)}$ in at most one row.

Throughout, for a matrix M , we let $M_{i,:}$ denote the i 'th row of M . The symbol ι denotes $\sqrt{-1}$. For an integer a , notation $[a]$ is shorthand for the set $\{1, \dots, a\}$. All logarithms are assumed to be base 2, unless the base is explicit, as in $\log_5 6$. For a matrix M over the field of real or complex numbers, $\|M\|$ is the spectral norm.

4 Replacing constant gates with an indeterminate

Fix an in-place algorithm $\mathcal{A} = (M^{(0)} = \text{Id}, M^{(1)}, \dots, M^{(m)} = F)$ computing F . Let $\mathcal{C}_{\mathcal{A}}$ denote the set of values used in the constant gates. Let $0 < \Delta \leq 11$ be some real number. If each $c \in \mathcal{C}_{\mathcal{A}}$ is an integral power of Δ , then we say that \mathcal{A} is integral with respect to Δ . We will assume that $\Delta \geq 2/3$ (otherwise we could replace Δ with $\Delta^{1/\ell}$ for a suitable integer ℓ , with respect to which \mathcal{A} is still integral).

Assume that \mathcal{A} is integral with respect to some $2/3 \leq \Delta \leq 1$. (In what follows, we shall make this assumption, but remove it in the appendix using limit arguments.) We define a new algorithm \mathcal{A}_{Δ} with corresponding matrices $(M_{\Delta}^{(0)}, \dots, M_{\Delta}^{(m)})$ where each $M_{\Delta}^{(t)}$ is now an $n \times n$ matrix over the ring of Laurent polynomials with complex coefficients in an indeterminate z .¹ (Throughout, by *polynomials* we refer to Laurent polynomials, and by *proper polynomials* we mean that only nonnegative power monomials are allowed). To get accustomed to our notation, note that for some matrix $A = A[z]$ in this ring we can write e.g. $A_{1,3}[7]$, referring to the (complex) number in the first row, third column of the matrix obtained by assigning the value 7 to z in $M_{\Delta}^{(t)}$.

We now inductively define $M_{\Delta}^{(t)}$. The first matrix $M_{\Delta}^{(0)}$ is simply $M^{(0)} = \text{Id}$. Having defined $M_{\Delta}^{(t-1)}$, we define $M_{\Delta}^{(t)}$ depending on whether the t 'th gate is a rotation, or a constant

¹ We remind the reader that a Laurent polynomial possibly has negative degree monomials.

gate. In case of rotation, the matrix $M_{\Delta}^{(t)}$ is obtained from $M_{\Delta}^{(t-1)}$ by performing the same transformation giving $M^{(t)}$ from $M^{(t-1)}$. More precisely, if $M^{(t)}$ is obtained from $M^{(t-1)}$ by a 2×2 rotation matrix R , namely $M^{(t)} = RM^{(t-1)}$, then $M_{\Delta}^{(t)} = RM_{\Delta}^{(t-1)}$.

In case of a constant gate with value $0 < c \in \mathcal{C}_{\mathcal{A}}$, we replace the value c with the monomial $z^{\log_{\Delta} c}$. More precisely, if $M^{(t)}$ is obtained from $M^{(t-1)}$ by left-multiplication with the matrix $\text{diag}(1, \dots, 1, c, 1, \dots, 1)$, then

$$M_{\Delta}^{(t)} = \text{diag}(1, \dots, 1, z^{\log_{\Delta} c}, 1, \dots, 1) M_{\Delta}^{(t-1)}.$$

By the integrality of \mathcal{A} with respect to Δ , the number $\log_{\Delta} c$ is indeed an integer. From the definition of $M_{\Delta}^{(t)}[z]$, a simple induction implies that for all $t \in \{0 \dots m\}$,

$$M_{\Delta}^{(t)}[\Delta] = M^{(t)}.$$

In words, this means that the t 'th matrix in the original algorithm \mathcal{A} is obtained by polynomial evaluation at Δ .

Throughout, for a polynomial $p = p[z]$, we let $\text{coeff}_i(p)$ denote the coefficient (matrix, vector or scalar) of p corresponding to z^i . In particular, $p = \sum_i \text{coeff}_i(p) z^i$.²

4.1 Paraunitary Matrices and Algebraic Condition Number

Here and throughout, for a complex number u , \bar{u} is complex conjugation. For a complex polynomial $p = p[z]$, complex conjugation \bar{p} is the polynomial obtained by conjugating the coefficients of p and substituting z^{-1} for z . For a polynomial matrix $M = M[z]$, the matrix M^* is obtained by transposing M and applying *polynomial* complex conjugation (as just defined) to all its elements. For a nonzero polynomial $p[z]$, let $\deg(p)$ denote the maximal degree of z (with a nonzero coefficient), and let $\text{val}(p)$ denote the minimal degree. (The operator name val is short for *valuation*, a standard algebraic term for *minimal degree* in our setting.) For example, if $p[z] = -6z^{-2} + 3 + z^5$, then $\deg(p) = 5$, $\text{val}(p) = -2$. Note that $\deg(p)$ may be negative, and $\text{val}(p)$ may be positive. Abusing notation, for a polynomial matrix $M[z]$, let $\deg(M) = \max_{i,j} \deg(M_{i,j})$ and $\text{val}(M) = \min_{i,j} \text{val}(M_{i,j})$.

► **Definition 2.** A complex polynomial matrix $U[z]$ is *paraunitary* if $U^*U = \text{Id}$.

Paraunitary matrices are used in signal processing (see e.g. [13]), and have some useful properties for our purposes. First, they form a multiplicative group. This fact is useful for verifying the following:

► **Claim 3.** The matrices $M_{\Delta}^{(t)}$ are paraunitary for all t .

The following is easy to see from the definitions, using simple induction:

► **Claim 4.** For paraunitary $U[z]$, the evaluation $U[\omega]$ is a (complex) unitary matrix for any ω a complex number on the complex unit circle.

This will be used below. We now remind the reader of the definition of a κ -well conditioned algorithm [2, 3]. In this work, we will refer to this standard notion of well conditionedness *geometric*, because it is related to geometric stretching of vectors under linear operators.

² We avoid subscripting for denoting polynomial coefficients, because that would be confused with matrix or vector indexing.

► **Definition 5.** The (geometric) condition number of a nonsingular complex matrix M is $\|M\| \cdot \|M^{-1}\|$. A nonsingular matrix M is (geometrically) κ -well conditioned if its (geometric) condition number is at most κ . Otherwise it is (geometrically) κ -ill conditioned. The algorithm $\mathcal{A} = (\text{Id}, M^{(1)} \dots M^{(m)})$ is *geometrically* κ -well conditioned if $M^{(t)}$ is geometrically well conditioned for all $t = 0 \dots m$. Otherwise, it is κ -ill conditioned. The (geometric) condition number of \mathcal{A} is the smallest κ such that \mathcal{A} is (geometrically) κ -well conditioned.

We define a different, though not unrelated (as we shall see below) notion of condition number of an algorithm. We first define it for Δ -integral algorithms, and then extend to general algorithms.

► **Definition 6.** A Δ -integral algorithm \mathcal{A} is Δ -*algebraically* κ -well conditioned if for all $t = 1 \dots m$, $\Delta^{\text{val}(M_{\Delta}^{(t)}) - \text{deg}(M_{\Delta}^{(t)})} \leq \kappa$. Otherwise, it is Δ -algebraically κ -ill conditioned.

Although the definition seems to depend on it, the parameter Δ is immaterial for defining (and thinking about) algebraic condition number. It is trivial to see that if \mathcal{A} is integral with respect to both Δ and Δ' and is Δ -algebraically κ -well conditioned, then it is also Δ' -algebraically κ -well conditioned. In fact, even if an algorithm is integral with respect to *no* Δ , we can extend the definition by using limits. Being algebraically κ -well conditioned is henceforth a property of any algorithm \mathcal{A} , regardless of integrality. The precise definitions are given in Appendix A, for completeness. We henceforth define:

► **Definition 7.** The algebraic condition number of \mathcal{A} is the smallest κ such that \mathcal{A} is algebraically κ -well conditioned.

If we consider some intermediary matrix $M^{(t)}$ given rise to by our algorithm \mathcal{A} , then algebraic well conditioning cannot be determined from $M^{(t)}$ ‘in vitro’, because it depends on *how* we reached $M^{(t)}$ from the initial state Id . This is in contrast with the notion of (geometric) condition, which can be determined from a single matrix. Nevertheless, the following lemma tells us that the two notions are quantitatively related.

► **Lemma 8.** *If an algorithm \mathcal{A} is algebraically κ -well conditioned, then it is geometrically κ -well conditioned. There exist algorithms for which the two notions of condition number coincide.*

The proof of this connection requires some complex analysis, and in particular the maximum modulus theorem. We provide it here only for the case \mathcal{A} is integral with respect to some Δ . The extension to general \mathcal{A} is not difficult using limit arguments, and we present it in Appendix A.2.

Proof. Assume \mathcal{A} is integral with respect to $\frac{2}{3} \leq \Delta \leq 1$, and fix $t \in [m]$. Let $v = \text{val}(M_{\Delta}^{(t)})$ and $d = \text{deg}(M_{\Delta}^{(t)})$. By definition of val and deg , the polynomial matrix $M^v[z] := z^{-v} M_{\Delta}^{(t)}$ is a proper polynomial matrix (with no negative degree monomials). Therefore, it is an entire (matrix valued) function of z , in the complex sense. By the (weak) maximum modulus theorem for Banach space valued functions on the complex plane ([7], page 230), we conclude that

$$\|M^v[\Delta]\| \leq \max_{\omega \in \mathbb{C}: |\omega|=1} \|M^v[\omega]\| ,$$

because Δ is contained in the open unit disk. The right hand side is identically 1 by Claims 3 and 4. The left hand side equals exactly $\Delta^{-v} \|M_{\Delta}^{(t)}[\Delta]\| = \Delta^{-v} \|M^{(t)}\|$. Therefore,

$$\|M^{(t)}\| \leq \Delta^v . \tag{2}$$

By Claim 3 and Definition 2, the inverse of the matrix $M_{\Delta}^{(t)}$ is $(M_{\Delta}^{(t)})^*$.³ Let

$$M^d := z^d (M_{\Delta}^{(t)})^* .$$

By construction, the polynomial matrix M^d is a proper polynomial, and hence again by the (weak) maximum modulus theorem,

$$\Delta^d \|(M_{\Delta}^{(t)})^*[\Delta]\| = \|M^d[\Delta]\| \leq \max_{\omega \in \mathbb{C}: |\omega|=1} \|M^d[\omega]\| = 1 .$$

But by paraunitarity $(M_{\Delta}^{(t)})^*$ is the inverse of $M_{\Delta}^{(t)}$, and this holds in particular for $z = \Delta$. Hence the last inequality gives $\Delta^d \|(M^{(t)})^{-1}\| \leq 1$ or:

$$\|(M^{(t)})^{-1}\| \leq \Delta^{-d} . \quad (3)$$

Combining (2) with (3), we get

$$\|(M^{(t)})^{-1}\| \cdot \|M^{(t)}\| \leq \Delta^v \cdot \Delta^{-d} \leq \kappa ,$$

where the rightmost inequality is by the lemma's assumption. This concludes the first part of the result.

To show tightness, it is enough to work with 2×2 matrices. Consider an algorithm \mathcal{A} performing two steps. The first multiplies the first coordinate by a positive number Δ , and the second multiplies the second coordinate by Δ^{-1} . It is easy to see that both the algebraic and the standard notions of condition number coincide for this case. ◀

We are now ready to state our main result.

► **Theorem 9.** *Assume algorithm $\mathcal{A} = (M^{(0)}, \dots, M^{(m)})$ computes F , and is algebraically κ -well conditioned. Then*

$$m = \Omega\left(\frac{n \log n}{\sqrt{\kappa}}\right) . \quad (4)$$

The proof is presented in the remainder of the paper, for the case \mathcal{A} is integral with respect to some $\frac{2}{3} \leq \Delta \leq 1$. The general case, using limit arguments, is deferred to Appendix A.3. We will use the notation $\rho := \frac{\log \kappa}{2 \log \Delta^{-1}}$, or $\Delta^{-\rho} = \sqrt{\kappa}$. We will also make the implicit assumption that $\kappa = n^{o(1)}$, because otherwise the statement of the theorem is obvious from the trivial observation that computation of F requires linearly many steps.

4.2 Polynomial Matrices: Norms and Entropy

We will need to define a norm for polynomials, polynomial vectors and polynomial matrices, over the complex field. For a polynomial $p = p[z] \in \mathbb{C}[z]$, we define

$$\|p\| = \sqrt{\sum_k |\text{coeff}_k(p)|^2} .$$

The following is well known from harmonic analysis:

$$\|p\| = \sqrt{\int_0^1 |p[e^{2\pi i t}]|^2 dt} . \quad (5)$$

³ The matrix $M_{\Delta}^{(t)}$ is real, and hence its transpose equals its conjugate-transpose. We prefer to keep the complex notation $(\cdot)^*$.

For a polynomial vector $v = v[z] \in \mathbb{C}[z]^n$ we define

$$\|v\| = \sqrt{\sum_{i=1}^n \|v_i\|^2}, \quad (6)$$

where we remind the reader that v_i is the i 'th coordinate of v . Finally, for a polynomial matrix A , we define $\|A\|_{2,\infty}$ as shorthand for $\max_i \|A_{i,:}\|$.

Let M denote a real paraunitary matrix. Then we extend the definition of the quasi-entropy potential from [3, 2] as follows:

$$\Phi(M) = \sum_k \sum_{i,j} h\left(|\text{coeff}_k(M_{i,j})|^2\right)$$

where $h : \mathbb{C} \mapsto \mathbb{C}$ is defined by $h(u) = -u \log |u|$. (Note for reviewers: No, this is not a mistake, and we do not want to define h as $-|u| \log |u|$. We will be using h for possibly negative and even complex values of u , where this makes a difference.⁴) For the Fourier transform F ,

$$\Phi(F) = \Theta(n \log n). \quad (7)$$

We also define a preconditioned version of Φ for any paraunitary matrix M , parameterized by two polynomial matrices A, B :

$$\Phi_{A,B}(M) = \sum_k \sum_{i,j} h\left(\text{coeff}_k((MA)_{i,j}) \overline{\text{coeff}_k((MB)_{i,j})}\right). \quad (8)$$

where the outer sum is over k for which the internal sum is not null, namely in the range

$$[\min\{\text{val}(MA), \text{val}(MB)\}, \max\{\deg(MA), \deg(MB)\}].$$

Note that $\Phi_{\text{Id},\text{Id}}(M) = \Phi(M)$. The general idea of a preconditioned quasi-entropy potential was developed in [2]. The idea there (and here) is to carefully choose fixed A and B , and to track the potential along the evolution of M under algorithm steps. The following key lemma bounds the absolute value of the change in entropy of a paraunitary matrix, incurred by multiplication of rows by monomials (corresponding to constant multiplication in the original algorithm) and by planar rotations (corresponding to rotations in the original algorithm).

► **Lemma 10.** (1) Let $M[z]$ be a paraunitary matrix, and let $A[z], B[z]$ be some complex matrices. Let $M'[z]$ be a paraunitary matrix obtained from $M[z]$ by multiplication by a diagonal matrix with monomials on the diagonal, that is

$$M'[z] = \text{diag}(z^{i_1}, \dots, z^{i_n})M[z].$$

Then

$$\Phi_{A,B}(M') = \Phi_{A,B}(M).$$

(2) If M' is obtained from M by a planar rotation action, then

$$|\Phi_{A,B}(M') - \Phi_{A,B}(M)| = O(\|MA\|_{2,\infty} \cdot \|MB\|_{2,\infty}).$$

⁴ This comment will be removed from published versions.

Proof. Part (1) is trivial, from the definitions. For part (2), polynomial matrix potential function is, equivalently, a scalar matrix potential function (defined in [3]) if we replace each polynomial by a row-vector of its coefficients. Therefore, using the potential change bound theorem of [3], we have the statement of the theorem. For self containment, we also provide a full proof in Appendix C. ◀

5 The preconditioners

We now define our two polynomial matrix preconditioners, A and B , that will be used to define a potential for proving the main theorem. For convenience, we will define $\mu = 1 - \Delta$, and let ℓ be the smallest integer such that $\Delta^\ell \leq 1/2$. By our assumptions on Δ , this also implies $\Delta^\ell \geq 1/4$, hence $\Delta^\ell = \Theta(1)$. Also note that

$$\ell = \Theta\left(\frac{1}{\mu}\right). \quad (9)$$

The preconditioner A is defined as $A[z] = \text{Id}(1 + \Delta z^{-1} + \Delta^2 z^{-2} + \Delta^3 z^{-3} + \dots + \Delta^{\rho+\ell} z^{-\rho-\ell})$. Consider now the polynomial matrix $P[z] = M_\Delta^{(m)}[z]A[z]$.

► **Claim 11.** The (matrix) coefficient of P corresponding to z^{-i} is exactly $F\Delta^i$ for all i in the range $R := \{\rho, \rho + 1, \dots, \rho + \ell\}$.

Proof. By polynomial multiplication definition, $\text{coeff}_{-i}(P)$ is

$$\begin{aligned} \sum_{k=-\rho-\ell}^0 \text{coeff}_{-i-k}(M_\Delta^{(m)}) \text{coeff}_k(A) &= \sum_{k=-\rho-\ell}^0 \Delta^{-k} \text{coeff}_{-i-k}(M_\Delta^{(m)}) \\ &= \Delta^i M_\Delta^{(m)}[\Delta] = \Delta^i F, \end{aligned} \quad (10)$$

as required. ◀

By our construction, for all i in the range R , $\Delta^i = \Theta(\Delta^\rho)$. Hence, we have that for all i in the range, $\text{coeff}_{-i}(M_\Delta^{(m)}[z]A[z])$ is $\Theta(\Delta^\rho)F = \Theta(\kappa^{-1/2})F$.

We now define the other preconditioner, B as $(M_\Delta^{(m)})^* \sum_{i=\rho}^{\rho+\ell} z^{-i} F$. We make a short note to demystify our choice of the two preconditioners. By the definition of B , $M_\Delta^{(m)}B = \sum_{i=\rho}^{\rho+\ell} z^{-i} F$. This implies that for all i in the range R , the coefficient corresponding to z^i of both $M_\Delta^{(m)}A$ and of $(M_\Delta^{(m)})^*B$ are proportional to F . Thus we have aligned the two operands of the product inside $h(\cdot)$ in (8) so that the potential is high. At the same time, in the beginning of the computation, the potential is low because A has only very few nonzeros (namely, on the diagonal). Creating this big potential gap, which we quantify in Lemma 13 below, is the one of the requirement for the proof to work. The other is upper bounding the potential change at each step, as we now do.

► **Lemma 12.** For any paraunitary M ,

$$\|MA\|_{2,\infty} \leq \Theta(\sqrt{\ell}), \quad \|MB\|_{2,\infty} \leq \Theta(\sqrt{\ell}).$$

Proof. Both matrices MA and MB belong to the family \mathcal{X} defined as

$$\mathcal{X} := \{U[z]p[z]X : U \text{ is paraunitary, } p \text{ is a polynomial in } \mathbb{C}[z], X \in \mathbb{C}^{n \times n} \text{ is unitary}\}.$$

We first show that for any $Y = U[z]p[z]X \in \mathcal{X}$, any row of Y has norm $\|p\|$. Fix $i \in [n]$, then

$$\begin{aligned}
 \|Y_{i,:}\|^2 &= \sum_{j=1}^n \|Y_{i,j}\|^2 = \sum_{j=1}^n \int_0^1 |Y_{i,j}[e^{2\pi i t}]|^2 dt = \sum_{j=1}^n \int_0^1 Y_{i,j}[e^{2\pi i t}] \overline{Y_{i,j}[e^{2\pi i t}]} dt \\
 &= \int_0^1 \sum_{j=1}^n Y_{i,j}[e^{2\pi i t}] \overline{Y_{i,j}[e^{2\pi i t}]} dt = \int_0^1 \sum_{j=1}^n (U[z]p[z]X)_{i,j}[e^{2\pi i t}] \overline{(U[z]p[z]X)_{i,j}[e^{2\pi i t}]} dt \\
 &= \int_0^1 \sum_{j=1}^n (U[z]p[z]X)_{i,j}[e^{2\pi i t}] \overline{(U[z]p[z]X)_{i,j}[e^{2\pi i t}]} dt \\
 &= \int_0^1 \left(\sum_{j=1}^n (U[z]p[z]X)_{i,j} \overline{(U[z]p[z]X)_{i,j}} \right) [e^{2\pi i t}] dt \\
 &= \int_0^1 (U[z]p[z]X)(U[z]p[z]X)^*_{i,i} [e^{2\pi i t}] dt = \int_0^1 (U[z]p[z]X X^* p^*[z]U^*[z])_{i,i} [e^{2\pi i t}] dt \\
 &= \int_0^1 (U[z]p[z]p^*[z]U^*[z])_{i,i} [e^{2\pi i t}] dt = \int_0^1 (p[z]p^*[z]) (U[z]U^*[z])_{i,i} [e^{2\pi i t}] dt \\
 &= \int_0^1 (p[z]p^*[z]) [e^{2\pi i t}] dt = \int_0^1 |p[e^{2\pi i t}]|^2 dt = \|p\|^2.
 \end{aligned}$$

The first equality was by definition, the second by (5), the fourth by changing order of integration and summation, the fifth by the definition of Y , the sixth by our definition of polynomial conjugation, the seventh by the fact that polynomial evaluation commutes with polynomial multiplication, the eighth by definition of matrix multiplication, the ninth by properties of conjugation of products, the tenth by unitarity of X , the eleventh by commutativity of p (a polynomial over scalars) and P (a polynomial over matrices), the twelfth by paraunitarity of U , and the fourteenth again by (5).

Now, by our construction, MA equals $U[z]p[z]X \in \mathcal{X}$ for $U = M$, $X = \text{Id}$, $p[z] = 1 + \Delta z^{-1} + \Delta^2 z^{-2} + \dots + \Delta^{\rho+\ell} z^{-\rho-\ell}$. Using (11),

$$\|A\|_{2,\infty} = \|p\| = \sqrt{\sum_{k=0}^{\rho+\ell} \Delta^{2k}} \leq \sqrt{\frac{1}{1-\Delta^2}} = \sqrt{\frac{1}{1-(1-\mu)^2}} = \Theta(\sqrt{1/\mu}) = \Theta(\sqrt{\ell}).$$

As for MB , notice that it equals $U[z]p[z]X \in \mathcal{X}$ for $U = M \cdot (M_{\Delta}^{(m)}[z])^*$, $p = \sum_{i=\rho}^{\rho+\ell} z^{-i}$, $X = F$. Therefore again we use (11) to obtain:

$$\|B\|_{2,\infty} = \|p\| = \sqrt{\sum_{k=\rho}^{\rho+\ell} 1} = \Theta(\sqrt{\ell}).$$

This concludes the proof. ◀

Let us now compute the preconditioned quasi-entropy of $M_{\Delta}^{(0)} = \text{Id}$ and of $M_{\Delta}^{(m)}$.

► **Lemma 13.**

$$\Phi_{A,B}(M_{\Delta}^{(m)}) = \Omega\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right) \quad \Phi_{A,B}(\text{Id}) = o\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right).$$

Proof. For the left bound in the lemma statement:

$$\begin{aligned}
\Phi_{A,B}(M_\Delta^{(m)}) &= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h \left(\text{coeff}_k((M_\Delta^{(m)} A)_{i,j}) \cdot \overline{\text{coeff}_k((M_\Delta^{(m)} B)_{i,j})} \right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h \left((\Delta^{-k} F_{i,j}) \cdot (F_{i,j}) \right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} h(F_{i,j}^2 \Delta^{-k}) = - \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} \Delta^{-k} F_{i,j}^2 \log(\Delta^{-k} F_{i,j}^2) .
\end{aligned} \tag{11}$$

Recall that $\Delta^{-\rho} = \kappa^{1/2} = n^{o(1)}$. Also recall that by definition of ℓ , $\Delta^\ell = \Theta(1)$. Therefore $\log \Delta^{-k}$ is $o(\log n)$ for all $k \in R$, and

$$\begin{aligned}
\Phi_{A,B}(M_\Delta^{(m)}) &= - \sum_{k=-\rho-\ell}^{-\rho} \sum_{i,j} (\Delta^{-k} F_{i,j}^2 \log F_{i,j}^2 + o(\log n) \Delta^{-k} F_{i,j}^2) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} \Phi(F) - o(\ell \Delta^\rho n \log n) = \Theta(\ell \Delta^\rho) \Phi(F) - o(\ell \Delta^\rho n \log n) \\
&= \Theta(\ell \Delta^\rho) n \log n - o(\ell \Delta^\rho n \log n) = \Omega(\ell \kappa^{-1/2} n \log n) ,
\end{aligned}$$

where the second equality is from $\sum F_{i,j}^2 = n$ (unitarity of F) and definition of Φ , the third is from the fact that $\Delta^i = \Theta(1)$ for all $i = 0 \dots \ell$, and the fourth from (7). For the right bound in the lemma statement:

$$\begin{aligned}
\Phi_{A,B}(\text{Id}) &= \sum_k \sum_{i,j=1}^n h \left(\text{coeff}_k((A)_{i,j}) \cdot \overline{\text{coeff}_k((B)_{i,j})} \right) \\
&= \sum_{k=-\rho-\ell}^{-\rho} \sum_{i=1}^n h \left((\Delta^{-k} 1) \cdot \overline{\text{coeff}_k(B_{i,i})} \right) = \sum_{i=1}^n \sum_{k=-\rho-\ell}^{-\rho} h \left((\Delta^{-k} 1) \cdot \overline{\text{coeff}_k(B_{i,i})} \right) .
\end{aligned}$$

Fixing $i \in [n]$, we upper bound the absolute value of the last inner sum, which equals:

$$\underbrace{\sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} h \left(\overline{\text{coeff}_k(B_{i,i})} \right)}_{E_1} - \underbrace{\sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} \overline{\text{coeff}_k(B_{i,i})} \log \Delta^{-k}}_{E_2} .$$

To bound E_1 , we first Lemma 12, to argue that for all $i \in [n]$, $\|B_{i,:}\|^2 = \ell + 1$. (To be precise, it was only shown that $\|B_{i,:}\|^2 = \Theta(\ell)$ but it is obvious from the proof of Lemma 12 that the exact value is $\ell + 1$.) This in particular implies that $\sum_{k=-\rho-\ell}^{-\rho} |\text{coeff}_k(B_{i,i})|^2 \leq \ell + 1$. We therefore find

$$\sup_{\sum \tau_k^2 \leq \ell+1} \left| \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} \tau_k \log |\tau_k| \right| . \tag{12}$$

This can be easily shown to be $\Theta(\kappa^{-1/2} \ell)$, using standard calculus, and using the fact that the coefficients Δ^{-k} are equal to each other (and to $\Delta^\rho = \kappa^{-1/2}$) up to a multiplicative global constant. We provide details in Appendix B. To bound E_2 , we use the fact that for all $k \in R$, $\Delta^{-k} = \Theta(\Delta^\rho) = \Theta(\kappa^{-1/2})$ and $\log \kappa = o(\log n)$. Therefore $|E_2|$ is $o \left(\kappa^{-1/2} (\log n) \sum_{k=-\rho-\ell}^{-\rho} |\text{coeff}_k(B_{i,i})| \right)$. But by the well known ℓ_1/ℓ_2 bound, we have that

$\sum_{k=-\rho-\ell}^{-\rho} |\text{coeff}_k(B_{i,i})| \leq \ell + 1$. Therefore, $|E_2|$ is $o(\kappa^{-1/2} \ell \log n)$. Combining, we have that $|\Phi_{A,B}(\text{Id})| = o(\ell \kappa^{-1/2} n \log n)$. \blacktriangleleft

Combining Lemma 13 and 10 with Claim 12, we get that the algorithm \mathcal{A} increases the potential defined by the preconditioned quasi-entropy $\Phi_{A,B}(\cdot)$ from $o\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right)$ to $\Omega\left(\frac{\ell n \log n}{\sqrt{\kappa}}\right)$, and at each step it can change the potential by no more than $\Theta(\sqrt{\ell})\Theta(\sqrt{\ell}) = \Theta(\ell)$. Hence, the number of steps is as stated in Theorem 9, concluding its proof.

6 Future Work

The main open question in this line of work is Conjecture 1. Since algebraic conditionedness is a weaker notion than typical (geometric) conditionedness, it should be expected that proving the conjecture with respect to the *algebraic* notion of condition number is easier:

► **Conjecture 14.** [Algebraic version] κ -Algebraic well conditioned computation allows only $O(\log(\kappa))$ speedup of FFT.

We now explain why this conjecture is very reasonable, using insights from this work. Recall that we started with an algorithm $\mathcal{A} = (\text{Id} = M^{(0)}, M^{(1)}, \dots, M^{(m)} = F)$ working in \mathbb{R}^n , and mapped it to a pseudo-algorithm \mathcal{A}_Δ working in n dimensions over the ring of polynomials, for the purpose of analysis. We then applied the resulting operator $M_\Delta^{(m)}$ outputted by \mathcal{A}_Δ to the preconditioner A by left-multiplication, yielding a polynomial matrix that is not far from being paraunitary, and with multiple copies of F scaled down by $\Theta(\sqrt{\kappa})$ as coefficients. This scaling down is the reason for the denominator we get in the bound of Theorem 9. Interestingly, very recently Ailon and Yehuda [5] were able to show that any algorithm computing a so-called “ ε -Perturbation of Fourier”, defined as $\text{Id} + \varepsilon F$ (for some small epsilon) in an almost perfectly conditioned algorithm must make at least $\Omega\left(\frac{n \log n}{\log(1/\varepsilon)}\right)$

steps.⁵ Moreover, the weaker bound $\Omega\left(\frac{n \log n}{\sqrt{1/\varepsilon}}\right)$, explained as a ‘warmup’ in [5], used ideas that are somewhat similar to the ideas in this paper. This leads us to believe that ideas possibly similar to those achieving the logarithmic dependence in $1/\varepsilon$ in [5] might be used to get logarithmic dependence on κ as stated in our conjecture. So far we have failed in these attempts.

It is interesting to study algebraic condition number in its own right as a measure of numerical stability. Consider the following related question: Lemma 8 tells us that if an algorithm is algebraically κ well conditioned then it is also (geometrically) κ -well conditioned. The converse is clearly not true, but perhaps some weak form of the converse should hold. In the extreme case, for example, note that an algorithm that is (geometrically) 1-well conditioned *must also be* algebraically 1-conditioned, because such an algorithm can only use constants 1 and -1 . Could we generalize this observation to obtain some kind of converse for Lemma 8?

Some more noteworthy open problems: (1) Extending the results to the extra-memory (non in-place) model. (2) Obtaining computational lower bounds for computing a Johnson-Lindenstrauss transform. There has been work showing that a JL transform can be sped up using the Fourier transform [4], but computational lower bounds for Fourier do not imply computational lower bounds for JL. Note that there are tight lower bounds for dimensionality parameters of JL [8], but they are not computational.

⁵ The paper [5] is also submitted to this conference.

References

- 1 Nir Ailon. A lower bound for fourier transform computation in a linear model over 2×2 unitary gates using matrix entropy. *Chicago J. of Theo. Comp. Sci.*, 2013.
- 2 Nir Ailon. Tighter fourier transform lower bounds. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 14–25, 2015.
- 3 Nir Ailon. An $\omega((n \log n)/R)$ lower bound for fourier transform computation in the r -well conditioned model. *TOCT*, 8(1):4, 2016.
- 4 Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual BCH codes. *Discrete & Computational Geometry*, 42(4):615–630, 2009.
- 5 Nir Ailon and Gal Yehuda. The complexity of computing a fourier perturbation. *CoRR*, abs/1604.02557, 2016. URL: <http://arxiv.org/abs/1604.02557>.
- 6 J. W Cooley and J. W Tukey. An algorithm for the machine computation of complex Fourier series. *J. of American Math. Soc.*, pages 297–301, 1964.
- 7 N. Dunford and J. Schwartz. *Linear Operators, Vol. I*. Interscience, New York, 1958.
- 8 Kasper Green Larsen and Jelani Nelson. The johnson-lindenstrauss lemma is optimal for linear dimensionality reduction. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 82:1–82:11, 2016.
- 9 Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- 10 Christos H. Papadimitriou. Optimality of the fast Fourier transform. *J. ACM*, 26(1):95–102, January 1979.
- 11 S. Winograd. On computing the discrete Fourier transform. *Proc. Nat. Assoc. Sci.*, 73(4):1005–1006, 1976.
- 12 S. Winograd. Some bilinear forms whose multiplicative complexity depends on the field of constants. *Math. Systems Theo.*, 10(169), 1976.
- 13 J. Zhou, MN Do, and J Kovacevic. Special paraunitary matrices, Cayley transform, and multidimensional orthogonal filter banks. *IEEE Trans. Image Process.*, 15(2):511–519, Feb 2006.

A

The General Case: Algorithms That are Not Integral

In this section we show how to extend the main result in this work to the case in which the algorithm \mathcal{A} is integral with respect to no Δ .

A.1 Extension of ‘Well Conditioned’ Definition to the General Case

Fix an algorithm $\mathcal{A} = (\text{Id} = M^{(0)}, \dots, M^{(m)})$ in \mathbb{R}^n . Let

$$\Delta_1 < \Delta_2 < \dots < \Delta_q < \dots$$

be an infinite, increasing sequence of numbers in the open interval $(\frac{2}{3}, 1)$, tending to 1 in the limit. For each index q we define another algorithm $\mathcal{A}_q = (M_q^{(0)}, \dots, M_q^{(m)})$ inductively, as follows. First, $M_q^{(0)} = \text{Id}$. For $t \geq 1$, $M_q^{(t)}$ is obtained from $M_q^{(t-1)}$ depending on whether $M^{(t)}$ is obtained from $M^{(t-1)}$ by a rotation or a constant gate. In case of rotation, $M_q^{(t)}$ is obtained by applying the same rotation to $M_q^{(t-1)}$. In case of a constant gate c acting on row i , The matrix $M_q^{(t)}$ is obtained from $M_q^{(t-1)}$ by multiplying the i ’th row by

$$c_q := \Delta_q^{\lceil \log_{\Delta_q} c \rceil}.$$

The resulting algorithm \mathcal{A}_q is clearly integral with respect to Δ_q . Additionally, it is not hard to see by standard limit calculus and by induction, that for each $t = 0 \dots m$, $\lim_{q \rightarrow \infty} M_q^{(t)} = M^{(t)}$. We are now ready to extend Definition 6 to the non integral case.

► **Definition 15.** \mathcal{A} is algebraically κ -well conditioned if for some sequence $\Delta_1 < \Delta_2 < \dots$ as above, there exists a sequence $\kappa_1, \kappa_2, \dots$ tending to κ , such that \mathcal{A}_q is κ_q -well conditioned for all q (per Definition 6).

It should be noted that if \mathcal{A} is algebraically κ -well conditioned, then any sequence $\Delta_1 < \Delta_2 < \dots$ (and not just one) will serve as a witness for the definition. It should also be noted that Definition 15 coincides with Definition 6 for algorithms \mathcal{A} that are Δ -integral for some Δ . This can be seen, for example, by choosing $\Delta_q = \Delta^{1/q}$ and $\kappa_q = \kappa$ for all q .

A.2 Proof of Lemma 8 for The General Case

Assume $\mathcal{A} = (M^{(0)}, \dots, M^{(m)})$ is algebraically κ -well conditioned (per the general Definition 15). Let $\Delta_1 < \Delta_2 < \dots$ and $\kappa_1, \kappa_2, \dots$ be as guaranteed to exist by the definition. Then by Lemma 8 (established for the integral case), the algorithm \mathcal{A}_q is geometrically κ_q -well conditioned for all q . But the geometric condition number of a matrix is a continuous invariant in the domain of nonsingular matrices, because both spectral norm and matrix inverse are continuous functions over matrices. Therefore, for all $t = 0 \dots m$, $M^{(t)}$ (which is also the limit of $M_q^{(t)}$) is geometrically κ -well conditioned.

A.3 Proof of Theorem 9 for the General Case

Assume $\mathcal{A} = (M^{(0)}, \dots, M^{(m)})$ is algebraically κ -well conditioned (per the general Definition 15). Let $\Delta_1 < \Delta_2 < \dots$ and $\kappa_1, \kappa_2, \dots$ be as guaranteed to exist by the definition. Let $F_q = M_q^{(m)}$, namely, the resulting matrix computed by \mathcal{A}_q . First we note that $\lim_{F_q \rightarrow F} F = F$. For each q , we now want to use the statement of the theorem in the integral case, which we have proved. The problem is that the resulting matrix F_q does not necessarily have the two key properties of Fourier transform used in the theorem: (1) unitarity and (2) high potential $\Phi(F_q)$.

In order to adjust the proof for the case of Δ_q -integral algorithm \mathcal{A} computing F_q (instead of F), we will first assume that q is large enough so that

$$\forall i, j \in [n] \quad |(F_q)_{i,j} - F_{i,j}| \leq |F_{i,j}|/2. \quad (13)$$

This is achievable, because $F_{i,j} \neq 0$ for all i, j , for both types of Fourier transform.⁶ In particular, (13) implies:

$$\sum F_{i,j}(F_q)_{i,j} = \Theta(n) \quad \Phi_F(F_q) := - \sum F_{i,j}(F_q)_{i,j} \log(F_{i,j}(F_q)_{i,j}) \geq \Omega(n \log n). \quad (14)$$

The trick is now to use preconditioners A_q, B_q as defined in Section 5.

$$A_q = \text{Id}(1 + \Delta_q z^{-1} + \Delta_q^2 z^{-2} + \Delta_q^3 z^{-3} + \dots + \Delta_q^{\rho_q + \ell_q} z^{-\rho_q - \ell_q}) \quad B_q = ((M_q)_{\Delta_q}^{(m)})^* \sum_{i=\rho_q}^{\rho_q + \ell_q} z^{-i} F,$$

⁶ We could also make due with zero valued matrix elements, but for simplicity we avoid this technicality here.

where ρ_q is $\frac{\log \kappa_q}{2 \log \Delta_q^{-1}}$ and ℓ_q is the smallest integer such that $\Delta_k^\ell \leq 1/2$. Note that in the definition of B_q we used F and not F_q . This is important, because we need Claim 12 and Lemma 13 to work (they rely on B being a product of a real unitary matrix and another paraunitary matrix). Lemma 13 now reads as

$$\Phi_{A_q, B_q}((M_q)_\Delta^{(m)}) = \Omega\left(\frac{\ell_q n \log n}{\sqrt{\kappa_q}}\right) \quad (15)$$

$$\Phi_{A_q, B_q}(\text{Id}) = o\left(\frac{\ell n \log n}{\sqrt{\kappa_q}}\right) \quad (16)$$

For the proof of Lemma 13, the only thing worth noting is that the expression in line (11) becomes

$$\sum_{k=-\rho_q-\ell_q}^{-\rho_q} \sum_{i,j} h((\Delta_q^{-k}(F_q)_{i,j}) \cdot (F_{i,j})) ,$$

that is, one copy of $F_{i,j}$ (coming from the preconditioner A_q) is replaced by $(F_q)_{i,j}$, and the other remains untouched. The remainder of the proof of the lemma proceeds in an obvious way, taking advantage of (14).

The resulting lower bound for the number m of steps that the algebraically κ_q -well conditioned algorithm \mathcal{A}_q needs to compute F_q is

$$C \frac{\Phi_F(F_q)}{\sqrt{\kappa_q}} ,$$

for sufficiently large q , for a global C . Taking q to infinity and recalling that $\kappa_q \rightarrow \kappa$ we conclude Theorem 9's proof for the general case.

B Missing Detail for Proof of Lemma 13

We upper bound (12): We first use the triangle inequality to upper bound (12) by

$$\sup_{\sum \tau_k^2 \leq \ell+1} \sum_{k=-\rho-\ell}^{-\rho} \Delta^{-k} |\tau_k \log |\tau_k|| . \quad (17)$$

We then use the fact that $\Delta^k = \Theta(\Delta^\rho) = \Theta(\kappa^{-1/2})$ for all $k \in R = \{\rho, \dots, \rho + \ell\}$. Therefore (17) is at most a constant times

$$\kappa^{-1/2} \sup_{\sum \tau_k^2 \leq \ell+1} \sum_{k=1}^{\ell+1} |\tau_k \log |\tau_k|| . \quad (18)$$

The contribution coming from k 's such that $|\tau_k| \leq 1$ to (18) is at most $\Theta(\kappa^{-1/2}\ell)$, because $|\tau \log |\tau||$ is a continuous function and hence bounded in the range $[0, 1]$. We hence bound

$$\sup_{K \leq \ell+1} \sup_{\left[\begin{array}{l} \forall k \leq K : \tau_k \geq 1 \\ \sum \tau_k^2 \leq \ell+1 \end{array} \right]} \kappa^{-1/2} \sum_{k=1}^K \tau_k \log \tau_k . \quad (19)$$

Using the method of lagrange multipliers, the inner supremum can only be obtained at one of the two points: (1) At the extreme candidate point ($\tau_1 = \sqrt{\ell - K + 2}$, $\tau_2 = \dots = \tau_K = 1$), for which the function value is $O(\kappa^{-1/2}\sqrt{\ell} \log \ell)$, or (2) when ($\tau_1 = \dots = \tau_K = \sqrt{\frac{\ell+1}{K}}$), for

which case the value is $\kappa^{-1/2} \sqrt{K(\ell+1)} \log \sqrt{\frac{\ell+1}{K}}$. But the last expression, maximized over $K \in [1, \ell+1]$, is at most $\Theta(\ell)$, obtained at $K = \Theta(\ell)$.

Combining these arguments, the value of (17) is $\Theta(\kappa^{-1/2}\ell)$, as required.

C Proof of Lemma 10

Without loss of generality, assume the rotation is applied to rows 1 and 2 of M , so

$$M' = \begin{pmatrix} \cos \theta & \sin \theta & & \\ -\sin \theta & \cos \theta & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{pmatrix} M \quad (20)$$

for some $\theta \in [0, 2\pi)$.⁷ It suffices to consider only the change in the contribution of rows 1 and 2 to the potential. The contribution to the pre-rotation potential coming from the first two rows before the rotation, which we denote $\Phi_{1,2}$, is

$$\Phi_{1,2} = \sum_k \sum_{j=1}^n \sum_{i=1}^2 h \left(\text{coeff}_k((MA)_{i,j}) \overline{\text{coeff}_k((MB)_{i,j})} \right).$$

Similarly, the contribution to the post-rotation potential coming from the first two rows, denoted $\Phi'_{1,2}$, is

$$\Phi'_{1,2} = \sum_k \sum_{j=1}^n \sum_{i=1}^2 h \left(\text{coeff}_k((M'A)_{i,j}) \overline{\text{coeff}_k((M'B)_{i,j})} \right).$$

By the triangle inequality:

$$|\Phi_{1,2} - \Phi'_{1,2}| \leq \sum_k \sum_{j=1}^n \left| \underbrace{\sum_{i=1}^2 h \left(\text{coeff}_k((MA)_{i,j}) \overline{\text{coeff}_k((MB)_{i,j})} \right) - h \left(\text{coeff}_k((M'A)_{i,j}) \overline{\text{coeff}_k((M'B)_{i,j})} \right)}_{\delta_{k,j}} \right|.$$

To avoid clutter, let

$$\alpha_{k,i,j} = \text{coeff}_k((MA)_{i,j}), \beta_{k,i,j} = \text{coeff}_k((MB)_{i,j}), \alpha'_{k,i,j} = \text{coeff}_k((M'A)_{i,j}), \beta'_{k,i,j} = \text{coeff}_k((M'B)_{i,j}).$$

For each k, j , let

$$\rho_{k,j} := \sqrt{|\alpha_{k,1,j}|^2 + |\alpha_{k,2,j}|^2} = \sqrt{|\alpha'_{k,1,j}|^2 + |\alpha'_{k,2,j}|^2} \quad (21)$$

$$\sigma_{k,j} := \sqrt{|\beta_{k,1,j}|^2 + |\beta_{k,2,j}|^2} = \sqrt{|\beta'_{k,1,j}|^2 + |\beta'_{k,2,j}|^2}, \quad (22)$$

⁷ To be technically precise, we also need to take into consideration reflections. But these can be composed as a product of a matrix as in (20) and a matrix with ± 1 on the diagonal. But it is trivial that such a matrix makes no difference to the potential.

where the right hand equality in both (21) and (22) is from the fact that a rotation preserves inner products (and in particular norms) in \mathbb{C}^2 . By the definition of h ,

$$\begin{aligned} h(\alpha_{k,1,j}\overline{\beta_{k,1,j}}) &= -\alpha_{k,1,j}\overline{\beta_{k,1,j}} \log |\alpha_{k,1,j}\overline{\beta_{k,1,j}}| \\ &= \alpha_{k,1,j}\overline{\beta_{k,1,j}} \log \left| \rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right| \\ &= \underbrace{-\alpha_{k,1,j}\overline{\beta_{k,1,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)} \end{aligned}$$

where we simply multiplied and divided by $\rho_{k,j}\sigma_{k,j}$ inside the logarithm to get the second line, then used logarithm properties and multiplied and divided by $\rho_{k,j}\sigma_{k,j}$ to get the third. Similarly:

$$\begin{aligned} h(\alpha_{k,2,j}\overline{\beta_{k,2,j}}) &= \underbrace{-\alpha_{k,2,j}\overline{\beta_{k,2,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)} \\ h(\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}) &= \underbrace{-\alpha'_{k,1,j}\overline{\beta'_{k,1,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)} \\ h(\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}) &= \underbrace{-\alpha'_{k,2,j}\overline{\beta'_{k,2,j}} \log |\rho_{k,j}\sigma_{k,j}|}_{(*)} \underbrace{-\rho_{k,j}\sigma_{k,j} \frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \log \left| \frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}} \right|}_{(**)} \end{aligned}$$

Now to estimate $\delta_{k,j} = h(\alpha_{k,1,j}\overline{\beta_{k,1,j}}) + h(\alpha_{k,2,j}\overline{\beta_{k,2,j}}) - h(\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}) - h(\alpha'_{k,2,j}\overline{\beta'_{k,2,j}})$, we first note that the contribution coming from the $(*)$ expressions vanishes. Indeed, rotation preserves complex inner product, and hence

$$\alpha_{k,1,j}\overline{\beta_{k,1,j}} + \alpha_{k,2,j}\overline{\beta_{k,2,j}} = \alpha'_{k,1,j}\overline{\beta'_{k,1,j}} + \alpha'_{k,2,j}\overline{\beta'_{k,2,j}}.$$

To estimate the contribution coming from the $(**)$ expressions, we simply note that the four fractions $\frac{\alpha_{k,1,j}\overline{\beta_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha_{k,2,j}\overline{\beta_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha'_{k,1,j}\overline{\beta'_{k,1,j}}}{\rho_{k,j}\sigma_{k,j}}$, $\frac{\alpha'_{k,2,j}\overline{\beta'_{k,2,j}}}{\rho_{k,j}\sigma_{k,j}}$ are at most 1 in absolute value. This implies that $|\delta_{k,j}| = O(\rho_{k,j}\sigma_{k,j})$. Summing up over all k and j , we get

$$\begin{aligned} \sum_{k,j} |\delta_{k,j}| &\leq O\left(\sum_{k,j} \rho_{k,j}\sigma_{k,j}\right) \leq O\left(\sqrt{\left(\sum_{k,j} \rho_{k,j}^2\right)\left(\sum_{k,j} \sigma_{k,j}^2\right)}\right) \\ &= O\left(\sqrt{(\|MA\|_{1,:})^2 + (\|MA\|_{2,:})^2} (\|(MB)_{1,:}\|^2 + \|(MB)_{2,:}\|^2)\right) \\ &= O(\|MA\|_{2,\infty} \cdot \|MB\|_{2,\infty}). \end{aligned}$$

This concludes the proof.